Méthodes numériques en Python

Rappels et définitions

R. Flamary

12 septembre 2019

Méthodes numériques en Python

Objectifs du Cours

- Utilisation pratique du langage Python (version 3).
- Introduction aux méthodes numériques.
- Implémenter des méthodes complexes à partir d'opérations de base.
- Initiation à la complexité algorithmique.

Méthodes numériques

- Séries entières.
- Équations non linéaires.
- Algèbre linéaire.
- Intégration et dérivation numérique.
- Interpolation polynomiale.

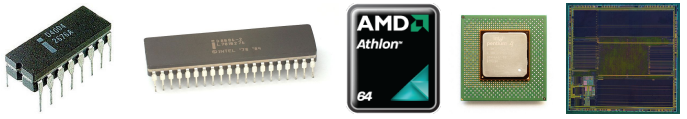
Plan du cours

Représentation des valeurs numériques

Principe

- Les nombres sont représentés en mémoire sur des bits.
- Le nombre de bits dédiés à chaque nombre définit ses limites.
- Les limites sont de natures différentes selon que l'on encode des nombres entiers ou réels.

Architectures



4 bits Intel 4004 (1971)

8 bits MOS Technology 6502 (NES 1985), Sharp x80 (GameBoy 1989)

16 bits Intel 8086 (1978), 65C816 (Super Nintendo 1990)

32 bits Intel x86 (1985), AMD Athlon K5 (1995), R3000A (Playstation 1994)

64 bits Athlon 6 (2003), Pentium 4 (2004), ARMv8-A (2011)

Nombres entiers

Entiers signés

Stocké en mémoire en base 2 :

$$x = s \sum_{i=0}^{p-2} d_i 2^i$$

- s signe (en binaire le signe moins est encodé par 1)
- p entier est le nombre de bits (souvent puissance de 2)
- $d_i \forall i \in 0, \ldots, p-2$ est un nombre binaire (0 ou 1).
- Représentation en mémoire (16 bits) :

signe	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
s	d_{14}	d_{13}	d_{12}	d_{11}	d_{10}	d_9	d_8	d_7	d_6	d_5	d_4	d_3	d_2	d_1	d_0

Exercice 1

Donner la représentation binaire des nombres suivants :

x	signe	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
5																
-10																
37																

Nombres entiers en Python

Type `int`

- Type entier de base en Python (objet).
- Tous les calculs sont sans pertes (pas de limite de taille).
- Taille mémoire dépendant de la valeur (attention à la complexité).
- La méthode `bit_length()` retourne le nombre de bits nécessaires au stockage de la variable.

Code source

```
1 i=100
2 print('i =',i)
3 print('type(i) =',type(i))
4 print('taille = ',i.bit_length(),'
      bits')
5 j=100**200 # 100^200
6 print('j =',j)
7 print('type(j) =',type(j))
8 print('taille = ',j.bit_length(),'
      bits')
```

Sortie

```
1 $python3 ex_int.py
2 i = 100
3 type(i) = <class 'int'>
4 taille = 7 bits
5 j =
      1000000000000000000000000000000000
6 type(j) = <class 'int'>
7 taille = 1329 bits
```


Nombres entiers Numpy

Bits	Usage	Intervalle	Déc.
8	`np.int8`	-128 à 127	3
16	`np.int16`	-32 768 à 32 767	5
32	`np.int32`	-2 147 483 648 à 2 147 483 647	10
64	`np.int`,`np.int64`	-9 223 372 036 854 775 808 à 9 223 372 036 854 775 807	19

- On suppose que numpy a été importé avec : `import numpy as np`
- Contrôle de la taille mémoire et du type.
- Les objets numpy retournent leur taille en octet avec la propriété `nbytes`.
- Attention lors des calculs numpy peux convertir automatiquement le type.

Entiers non signés

- Même intervalle mais commençant à 0.
- Déclarés en précédant l'identifieur de u comme `np.uint16`.

Exemple de déclaration

```
1 import numpy as np
2 i=np.int8(10)
3 j=np.int(-123456)
4 k=np.uint32(32)
```


Limitation des entiers Numpy

Taille et intervalle entier sur p bits

- Signé entre -2^{p-1} et $2^{p-1}-1$, non signé entre 0 et 2^p-1
- La fonction numpy `np.iinfo` retourne les limites d'une variable ou d'un type.
- Représentation exacte et règles arithmétiques respectées dans l'intervalle.
- Conversion automatique vers le plus grand si types différents.
- RuntimeWarning en cas de dépassement de l'intervalle.

Code source

```
1 import numpy as np
2 i=np.int8(120)
3 j=np.int8(10)
4 info=np.iinfo(i)
5 print(info.min, info.max)
6 print(i+10)
7 print(type(i+10))
8 print(i+j)
```

Sortie

```
1 $python3 ex_int_numpy.py
2 ex_int_numpy.py:8:
      RuntimeWarning: overflow
      encountered in byte_scalars
3   print(i+j)
4 -128 127
5 130
6 <class 'numpy.int64'>
7 -126
```

Nombres à virgule flottante

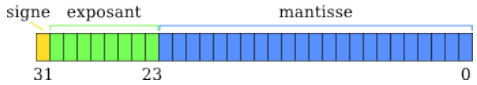
Virgule flottante

Un flottant est encodé comme :

$$x = sm2^{e-d}$$

- s signe
- $1 \leq m < 2$ mantisse (m)
- e exposant, d décalage de l'exposant

Norme IEEE 754



- Flottant 32 bits (s : 1 bits, m : 23 bits, e : 8 bits, d=$2^{8-1}-1$=127)
  - Chiffres significatifs : $2^{-23} \approx 1.10^{-7}$ $\rightarrow$ 7 chiffres
  - Min/max en valeur absolue : 1.10^{-38}, $3.4.10^{38}$
- Flottant 64 bits (s : 1 bits, m : 52 bits, e : 11 bits, d=$2^{11-1}-1$=1023)
  - Chiffres significatifs : $2^{-52} \approx 2.2.10^{-16}$ $\rightarrow$ 15 chiffres
  - Min/max en valeur absolue : $2.22.10^{-308}$, $1.79.10^{308}$

Opérations en virgule flottante

Soient x_1 et x_2 deux nombres flottants.

Addition

- L'addition nécessite que les deux nombres aient le même exposant.
- Si $e_1 = e_2$, on additionne les mantisses.
- Si $e_1 \neq e_2$, on décale la mantisse du nombre le plus petit pour qu'ils aient le même exposant, ensuite on additionne les mantisses.
- Si après l'addition la mantisse $m > 2$, on la décale vers la droite et on ajoute 1 à l'exposant.
- Attention si les nombres sont d'un ordre de grandeur très différents !

Multiplication

$$x_1 x_2 = m_1 m_2 2^{e_1+e_2-2d}$$

- On effectue le produit des mantisses et la somme des exposants.
- si m sort de l'intervalle, on décale la mantisse et change l'exposant.

Flottants en Python

Bits	Usage	max/min absolu	Déc.
32	np.float32	1.10^{-38}, $3.4.10^{38}$	7
64	np.float64	$2.22.10^{-308}$, $1.79.10^{308}$	15

- Type de base `float` dans Python et `np.float` dépendent du processeur.
- Numpy propose aussi le format `np.float16` moins précis mais efficace en mémoire.
- Les limites des types flottant sont retournées par la fonction `np.finfo`.
- La précision numérique et les erreurs d'arrondis doivent être pris en compte dans le codage.

Code source

```
1 x=np.float32(1.0)
2 print('x+1e-8={:1.15e}'.format(x+np.
      float32(1e-8)))
3 print('x+float32(1e50)={:1.15e}'.
      format(x+np.float32(1e50)))
4 print('x+1e50={:1.15e}'.format(x+1e
      -50))
5 y=np.float64(1.0)
6 print('y+1e-8={:1.15e}'.format(y+1e
      -8))
7 print('y+1e50={:1.15e}'.format(y+1e50
      ))
```

Sortie

```
1 $python3 ex_float.py
2 x+1e-8=1.000000000000000e+00
3 x+float32(1e50)=inf
4 x+1e50=1.000000000000000e+00
5 y+1e-8=1.000000010000000e+00
6 y+1e50=1.000000000000000e+50
```


Exemples d'encodage

Code source

```
1 lst=[3.14, 0.125, 9.3, 0.0, -0.0, np.nan, np.inf]
2 for f in lst:
3     print_float(f)
```

Sortie

```
1 $python3 ex_float_enc.py
2 3.140000 -> 0 10000000 10010001111010111000011
3 0.125000 -> 0 01111100 00000000000000000000000
4 9.300000 -> 0 10000010 00101001100110011001101
5 0.000000 -> 0 00000000 00000000000000000000000
6 -0.000000 -> 1 00000000 00000000000000000000000
7 nan -> 0 11111111 10000000000000000000000
8 inf -> 0 11111111 00000000000000000000000
```

Le langage Python 3 (rappels)

Python 3

- Langage de programmation interprété multi-plateforme.
- Typage dynamique, multi-paradigme (fonctionnel, object).
- Blocks de code séparés visuellement (espaces ou tab).
- Open source et gratuit, grande Bibliothèque standard.
- Bibliothèques dédié aux données (Numpy/Scipy/Matplotlib).
- De plus en plus utilisé en entreprise et pour la science des données.

Utiliser Python

- Installer sous Windows, MacOSX, Linux avec l'environnement Anaconda : `https://www.anaconda.com/distribution/`
- Pour exécuter un programme il faut lancer (sous linux) la commande : `python3 fichier.py`.
- De nombreux éditeurs peuvent être utilisés (gedit, idle3, vscode, spyder).
- Possibilité d'utiliser Jupyter Notebook pour des visualisations.

Chaines de caractères

- Ce sont des listes de caractères imprimables.
- Nombreuses fonctions de gestion définies dans le module `string`.
- Toujours possible de déterminer sa taille avec la fonction `len()`.
- Une chaîne de caractères est délimitée par ', " ou """.
- Le caractère \n représente un retour à la ligne, \t une tabulation.
- On accède aux caractères avec l'opérateur `[]` (indexage à 0).

Code source

```
c1='spam'
c2='eggs'
print(c1)
print(len(c1))
print(c1[2])
print(c1[2:])
print(c1[::2])
print(c1[::-1])
print(c1+' and '+c2)
```

Sortie

```
$python3 ex_string.py
spam
4
a
am
sa
maps
spam and eggs
```


Listes

Type `list`

- Type natif de Python, stocke une séquence d'objets.
- Création à partir de l'opérateur `[]` ou de la fonction `list()`.
- Aucune contrainte de type, indexage à base 0.
- Sélection d'éléments par slicing (opérateur `:`).
- Similaire au `tuple` qui utilise `()` mais ce dernier ne peut être modifié.
- Attention : ne pas utiliser pour stocker des vecteurs ou des matrices. Pas efficace en mémoire ou temps de calcul (voir cours algèbre linéaire).

Code source

```
lst=[0,1,2]
print(lst)
lst.append(3)
print(lst)
lst.insert(1,1.5)
print(lst)
lst[1]=2
print(lst)
print(lst[1])
print(lst[:2])
print(lst[2:])
```

Sortie

```
$python3 ex_list.py
[0, 1, 2]
[0, 1, 2, 3]
[0, 1.5, 1, 2, 3]
[0, 2, 1, 2, 3]
2
[0, 2]
[1, 2, 3]
```


Dictionnaires

Type `dict`

- Type natif de Python, stocke des relations clé/objets.
- Les object sont indéxés par des clés (peut être n'importe quel type).
- Création à partir de l'opérateur {}.
- Les clés peuvent être parcourues dans une boucle `for`.
- Un dictionnaire peut être passé à une fonction avec l'opérateur `**` (ex : template).

Code source

```
dic={'Bonjour':'Hello' , 'Monde' : '
    World'}
print(dic)
print(dic['Bonjour'])
dic[0]='spam'
dic[2]=45
print(dic)
for key in dic:
    print('key={},valeur="{}"'.format(
        key,dic[key]))
print('{Bonjour} {Monde} !'.format(**
    dic))
```

Sortie

```
$python3 ex_dict.py
{'Bonjour': 'Hello', 'Monde': '
    World'}
Hello
{'Bonjour': 'Hello', 'Monde': '
    World', 0: 'spam', 2: 45}
key=Bonjour,valeur="Hello"
key=Monde,valeur="World"
key=0,valeur="spam"
key=2,valeur="45"
Hello World !
```

Fonctions en Python

- Les fonctions permettent d'exécuter simplement une suite d'instructions (avec des paramètres).
- Elles sont définies avec le mot clé `def`.
- Le corp de la fonction est un block de code (délimité par tab ou 4 espaces) suivant la définition.
- Elles ont un nom et un/des paramètres d'entrée (optionnels si ils ont une valeur par défaut).
- Le mot clé `return` est utilisé pour retourner le résultat de la fonction.
- Peuvent être définie en une ligne avec le mot clé `lambda`.

Code source

```
1 def carre_plus(i,j=0):
2     return i*i+j
3 for i in range(3):
4     print('carre_plus(i) =',
          carre_plus(i))
5 for i in range(3):
6     print('carre_plus(i,i) =',
          carre_plus(i,i))
```

Sortie

```
1 $python3 ex_func.py
2 carre_plus(i) = 0
3 carre_plus(i) = 1
4 carre_plus(i) = 4
5 carre_plus(i,i) = 0
6 carre_plus(i,i) = 2
7 carre_plus(i,i) = 6
```


Exercice 2

Coder une fonction qui retourne le cube d'un nombre flottant :

$$cube(x) = x^3$$

Analyse du problème

- Nom : cube
- Entrée :
- Sortie :
- Mise en oeuvre :

Solution

Classes

Porgrammation object avec `class`

- Permet de définir de nouveaux objets (avec héritage).
- La fonction `__init__` définit l'initialisation de l'objet.
- Tous les opérateurs du langage peuvent être redéfinis (`+,-,*,/,print,(),[]`).
- Numpy et matplotlib utilisent des objets et leur opérateurs pour les calculs.

Code source

```
1 class MaClasse:
2     def __init__(self,nom='',prenom=''
          ):
3         self.nom=nom
4         self.prenom=prenom
5     def say_hello(self):
6         print('Hello {}!'.format(self.
              prenom))
7
8 wc=MaClasse('Churchill','Winston')
9 print(wc)
10 print(wc.nom)
11 wc.say_hello()
12 MaClasse().say_hello()
```

Sortie

```
1 $python3 ex_class.py
2 <__main__.MaClasse object at 0
      x7fcaf2e4bf28>
3 Churchill
4 Hello Winston!
5 Hello !
```


Modules en importation

- Une fonction peut être déclarée dans un autre fichier python (ou dossier) appelé module.
- Pour utiliser une fonction dans un module il faut l'importer avec `import`.
- Possibilité de renommer les modules ou fonctions lors de l'importation avec `as`.
- Import de fonction avec :`from module import fonction`.
- Pour pouvoir être importé les modules doivent être dans dossier courant ou un des dossier dans la liste `sys.path`.

Fichier `monmodule.py`

```
1 def carre(x):
2     """Calcule le
          carre
          de x"""
3     return x*x
4
5 def cube(x):
6     """Calcule le
          cube de
          x"""
7     return x*x*x
```

Fichier `ex_import.py`

```
1 import monmodule
2 import monmodule as mm
3 from monmodule import carre
4
5 print(monmodule.carre(2))
6 print(monmodule.cube(2))
7 print(mm.cube(2))
8 print(carre(2))
```

Sortie

```
1 $python3 ex_import.py
2 4
3 8
4 8
5 4
```

Sortie écran

Fonction `print` et formatage de chaine

- Imprime une chaîne de caractères à l'écran (terminal) suivi d'un retour à la ligne.
- Formater la chaînes de caractères avec la méthode `format` des chaînes de caractères.

Type	Format
Entier	{:d}
Entier non signé	{:u}
Flottant	{:f},{:e}
caractère	{:c}
Chaîne de caractère	{:s}

Code source

```
1 print('{:d}, {:e}, {:f}'.format
      (10,1.5,3.14))
2 print('{:05d}, {:1.5e}, {:1.2f}'.
      format(10,1.5,3.14))
3 print('{s1} and {s2}'.format(s2='eggs
      ',s1='spam'))
4 dic={'s1':'spam', 's2': 'eggs'}
5 print('{s1} and {s2}'.format(**dic))
```

Sortie

```
1 $python3 ex_print_format.py
2 10, 1.500000e+00, 3.140000
3 00010, 1.50000e+00, 3.14
4 spam and eggs
5 spam and eggs
```

Il est possible de demander à l'utilisateur de rentrer des informations avec fonction `input()` qui retourne une chaîne de caractères.

Entrée/Sortie par ligne de commande

- Des paramètres (séparés par des espaces) peuvent être donnés lors de l'exécution du programme.
- Les paramètres donnés en ligne de commande sont listés dans la liste `sys.argv` du module `sys`.
- Le premier paramètre (index 0) est toujours le nom du programme.
- Pour des configurations plus complexes le module `argparse` est fortement recommandé (aide automatique, conversion automatique de types).

Code source

```
1 import sys
2
3 print('Nombre de Params: {}'.format(
      len(sys.argv)))
4 for i,param in enumerate(sys.argv):
5     print('Param. {}: {}'.format(i,
          param))
```

Sortie

```
1 $python3 ex_command.py
2 Nombre de Params: 1
3 Param. 0: ex_command.py
4
5 $python3 ex_command.py 1 "Hello
      world"
6 Nombre de Params: 3
7 Param. 0: ex_command.py
8 Param. 1: 1
9 Param. 2: Hello world
```


Bibliothèque standard de Python

- Ce sont un ensemble de modules présents sur tous les systèmes.
- Liste des modules/fonctions les plus utiles, mais documentation détaillée sur le web.

Principaux modules (pour ce cours) :

`string`	Gestion des chaînes de caractères.
`math`	Fonctions mathématiques courantes.
`cmath`	Fonctions mathématiques courantes sur nombres complexes.
`sys`	Outils système (ligne de commande, informations)
`time,datetime`	Temps et affichage de dates.

Bibliothèque standard : `math`

Fonctions mathématiques de base :

floor	Arrondi en dessous.
ceil	Arrondi au dessus.
fabs	Valeur absolue.
exp	Exponentielle.
log, log10	Logarithme népérien et décimal.
pow	Puissance.
sqrt	Racine carrée.
sin, cos, tan	Fonctions trigonométriques.

Code source

```
1 import math
2 print('floor(3.14)',math.floor(3.14))
3 print('ceil(3.14)',math.ceil(3.14))
4 print('log(2)',math.log(2))
5 print('exp(1)',math.exp(1))
6 print('sqrt(2)',math.sqrt(2))
7 print('sin(2)',math.sin(1))
```

Sortie

```
1 $python3 ex_math.py
2 floor(3.14) 3
3 ceil(3.14) 4
4 log(2) 0.6931471805599453
5 exp(1) 2.718281828459045
6 sqrt(2) 1.4142135623730951
7 sin(2) 0.8414709848078965
```

Numpy

Bibliothèque Numpy

- Bibliothèque de calcul numérique, spécialisée pour les tableaux.
- Objets Python mais implémentation efficace en C/Fortran.
- Trés efficace et lisible pour l'algèbre linéaire.
- Formulation des calculs vectorisés (vecteurs/matrices).
- Import standard : `import numpy as np`

Code source

```
1 import numpy as np
2 v=np.arange(3)
3 print(v)
4 A=np.zeros((2,2))
5 A[0,0]=1
6 print(A)
7 print(A+2)
8 print(np.cos(A))
```

Sortie

```
1 $python3 ex_numpy.py
2 [0 1 2]
3 [[1. 0.]
4  [0. 0.]]
5 [[3. 2.]
6  [2. 2.]]
7 [[0.54030231 1. ]
8  [1. 1. ]]
```


Matplotlib

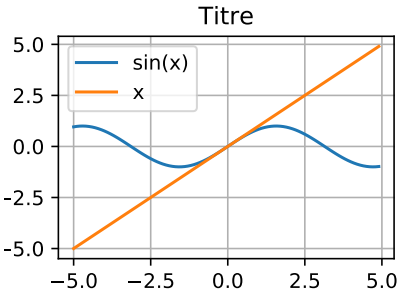
Bibliothèque `matplotlib,pylab`

- Bibliothèque de tracé/visualisation de figures.
- Tracé de courbes/nuages de points (`pl.plot`) et images (`pl.imshow`).
- Nombreux tutoriels sur le web.
- Import standard : `import pylab as pl` (des fois `plt` dans les exemples)

Code source

```
1 import numpy as np
2 import pylab as pl
3 x=.1*np.arange(100)-5
4 pl.figure(1,(3,2))
5 pl.plot(x,np.sin(x),label='sin(x
      )')
6 pl.plot(x,x,label='x')
7 pl.grid()
8 pl.legend()
9 pl.xlabel('x')
10 pl.title('Titre')
11 pl.savefig('ex_matplotlib.pdf')
```

Sortie (ex_matplotlib.pdf)



Algorithmique

- Un algorithme est une suite finie d'instructions permettant de donner la réponse à un problème.
- Il existe différentes représentations d'algorithmes (schéma, texte), dans ce cours nous donnerons des algorithmes sous forme de fonction Python.
- Pour résoudre un problème complexe on cherche à découper ce problèmes en plusieurs sous-problèmes plus simples.
- Un algorithme est défini par ses entrées, sorties et son nom.
- Les algorithmes utilisent typiquement des opérateurs de branchement (condition) et des boucles.

Fichier `carre.py`

```
1 def carre(x):
2     """Calcule le carre
          de x"""
3     s=x*x
4     return s
```

Algorithme correspondant

- Nom : **carre**
- Entrée : entier **i**
- Sortie : entier **s** (valeur $\mathbf{i}^2$)

Début

s ← **i*i**

Fin

Opérateurs logiques en Python

Ces opérateurs sont utilisés pour tester une condition booléenne (branchement ou boucle). Leur résultat peut être `False` ou `True`.

Comparaisons

`==, !=`	Égalité, inégalité.
`<,>`	Stricte comparaison.
`<=, >=`	Comparaison avec égalité

Opérations booléennes

`not`	Négation booléenne.
`and`	ET booléen.
`or`	OU booléen.
`^`	OU exclusif booléen.

Exercice 3

Donner la valeur logique retournée par ces expressions :

- `(1>=0) and not (10==10)` :
- `4==9/2` :
- `4==9//2` :
- `(1<=0) or not 10` :
- `i=3.; (i<=4) and (i>=1)` :
- `i=3.; (i>=4) and (i<=1)` :

Condition `if`

- Exécute une série d'instructions si une condition est vérifiée.
- Format standard : `if condition:`
- Le block de code suivant n'est exécuté quand quand la condition est vraie.
- Le mot clé `else` permet de définir des instructions exécutées si la condition est fausse :

```
if condition:
    instructions1
else:
    instructions2
```

- Il existe également un format en une ligne : `if condition: instruction`

Code source

```
a=3
if a>=5:
    print("a>=5")
else:
    print("a<5")
# Condition en une ligne
if a==3: print('Aussi a==3')
```

Sortie

```
$python3 ex_if.py
a<5
Aussi a==3
```


Condition `if/elif`

- Il n'existe pas d'opérateur `switch` en Python.
- `if/elif` permet de tester l'égalité d'une variable avec plusieurs valeurs.
- Commencer avec le mot clé `if` puis un `elif` pour chaque valeur.
- Le dernier cas `else` permet de gérer les cas qui n'ont pas été définis.

Code source

```
import sys
a=int(sys.argv[1])
if a in [1,2]:
    print('a est egale a 1 ou 2')
elif a==3:
    print('a est egale a 3')
else:
    print('Autre valeur')
```

Sortie

```
$python3 ex_ifelif.py 1
a est egale a 1 ou 2

$python3 ex_ifelif.py 3
a est egale a 3

$python3 ex_ifelif.py 5
Autre valeur
```


Boucle `for`

- Format en Python :

```
for i in iterable:
    instructions
```

- La boucle exécute les `instructions` pour toutes la valeurs de `i` retourné par l'itérable (liste, tuple ou générateur).
- Exemple d'itérations avec une variable entière :

Code source

```
for i in range(3): # i de 0 a 2
    s="i={}, i*i={}".format(i,i*i)
    print(s)
```

Sortie

```
$python3 ex_for.py
i=0, i*i=0
i=1, i*i=1
i=2, i*i=4
```

- Exemple d'itérations avec une liste :

Code source

```
for i in ['spam','eggs','pickles']:
    s="I want {}!".format(i)
    print(s)
```

Sortie

```
$python3 ex_for2.py
I want spam!
I want eggs!
I want pickles!
```


Boucle `while`

- Format en Python :

```
while condition:
    instructions
```

- La boucle exécute les `instructions` tant que `condition` est vraie.
- Exemple :

Code source

```
i=0
s=0
m=21
while s<m:
    print('i = {}, i*i = {}'.format
        (i,i*i))
    i+=1
    s=i*i
```

Sortie

```
$python3 ex_while.py
i = 0, i*i = 0
i = 1, i*i = 1
i = 2, i*i = 4
i = 3, i*i = 9
i = 4, i*i = 16
```

- Attention à l'initialisation et à bien vérifier que la boucle se termine.

Exercice 4 : Somme

Coder une fonction C calculant la valeur entière

$$sommecarre(n) = \sum_{i=0}^{n} i^2$$

Analyse du problème **Solution**

- Nom :
- Entrée :
- Sortie :
- Mise en oeuvre :

Exercice 5 : Produit

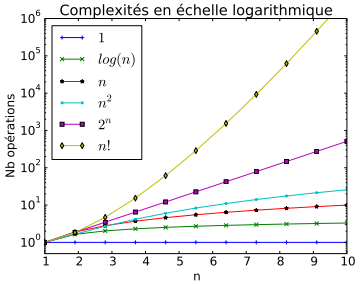
Coder une fonction C calculant la puissance n d'un flottant x :

$$puissance(x, n) = \prod_{i=1}^{n} x = x^n$$

Analyse du problème **Solution**

- Nom : `puissance`
- Entrée :
  - flottant `x`
  - entier `n`
- Sortie : flottant `f`
- Mise en oeuvre :
  - Création d'une variable d'accumulation.
  - Boucle `for`

Complexité d'un algorithme



Complexités typiques

Notation	Type
$O(1)$	constante
$O(log(n))$	logarithmique
$O(n)$	linéaire
$O(nlog(n))$	quasi-linéaire
$O(n^2)$	quadratique
$O(n^p)$	polynomiale
$O(2^n)$	exponentielle
$O(n!)$	factorielle

- La complexité en temps d'un algorithme est définie par le nombre d'opérations nécessaires à son exécution (Notation T(algo)).
- La complexité en espace d'un algorithme est définie par la taille mémoire nécessaire à son exécution (Notation S(algo)).
- Il est souvent difficile de connaître le nombre exact d'opérations et de la taille mémoire, on utilise donc la notation $O(f(n))$ qui permet de se concentrer sur les opérations/variables les plus complexes.
- n est une mesure de la taille des données (par exemple la taille d'un tableau ou le nombre de termes calculés dans une suite).

Notation $O(\cdot)$

- Cette notation permet d'illustrer la dominance d'une fonction par une autre.
- Soient f et g deux fonctions de la variable réelle x. On dit que f est dominée par g en $+\infty$, et on note

  $$f(x) = O(g(x))(x \to +\infty)$$

  lorsqu'il existe des constantes N et C telles que

  $$\forall x > N \qquad |f(x)| \leq C\,|g(x)|\,.$$

- Intuitivement, cela signifie que f ne croit pas plus vite que g.
- Utilisation pour borner asymptotiquement la complexité algorithmique.
- Aussi communément utilisée (parfois avec un petit o) pour les séries de Taylor :

$$e^x = 1 + x + \frac{1}{2}x^2 + O(x^2) \text{ quand } x \to 0$$

Calcul de complexité

- Les algorithmes de complexité déterministe ont un nombre d'opérations qui peut être déterminé à l'avance.
- Les programmes non déterministes ont un nombre d'opérations qui dépendent de manière complexe des données d'entrée.

Opération	Nom	Complexité $T()$
+,-,*,/	Opérations mathématiques	$O(1)$
=	Affectation	$O(1)$
`for i in range(n):`	Boucle `for`	$O(n)$
`if cond:`	Test `if`	$O(1)$

- Les complexités de boucles imbriquées se multiplient.
- La complexité d'une boucle `while` peut parfois être estimée (séries entières, vitesse de convergence).

Exercice 6 : Complexité

Calculer la complexité des fonctions suivantes :

Code

```
1 def carre_plus(i,j=0):
2     return i*i+j
```

Complexité

- Nb opérations :
- Complexité $T()$:
- Complexité $S()$:

Code

```
1 def sommecarree(n):
2     res=0
3     for i in range(n+1):
4         res+=i*i
5     return res
```

Complexité

- Nb opérations :
- Complexité $T()$:
- Complexité $S()$:

Code

```
1 def puissance(x,n):
2     res=x
3     for i in range(n-1):
4         res*=x
5     return res
```

Complexité

- Nb opérations :
- Complexité $T()$:
- Complexité $S()$:

Exercice 7 : Boucles imbriquées

Donner la complexité des fonctions suivantes visant à calculer :

$$func(x,n) = \sum_{i=1}^{n} x^i$$

Code

```
1 def puissance(x,n):
2     res=x
3     for i in range(n-1):
4         res*=x
5     return res
6
7 def func(x,n):
8     res=0
9     for i in range(1,n+1):
10        res+=puissance(x,i)
11    return res
```

Complexité

- Nb opérations :
- Complexité $T()$:
- Complexité $S()$:

Ressources bibliographiques I

[1] "Cours python 3 pour la programmation scientifique," https://courspython.com, version du 2019-08-28.

[2] "Standard python library," https://docs.python.org/3/library/, version du 2019-08-28.

[3] J. Bastien and J.-N. Martin, *Introduction à l'analyse numérique : applications sous Matlab : cours et exercices corrigés*. Dunod, 2003.

[4] D. Goldberg, "What every computer scientist should know about floating-point arithmetic," *ACM Computing Surveys (CSUR)*, vol. 23, no. 1, pp. 5–48, 1991.

[5] D. E. Knuth, *The Art of Computer Programming, Volume 1 (3rd Ed.) : Fundamental Algorithms*. Redwood City, CA, USA : Addison Wesley Longman Publishing Co., Inc., 1997.

[6] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical recipies in C*. Cambridge university press Cambridge, UK, 1992.